Header Processing Requirements and Implementation Complexity for IPv4 Routers

Andreas Moestedt*, Peter Sjödin*, Torsten Köhler†
Internet Research Institute
HP Laboratories Bristol
HPL-IRI-98-001
September, 1998

header processing,
IPV4, IPV6,
internet,
networking,
router performance,
packet classification

The complexity of IP header processing is commonly believed to be the main limiting factor for router performance. The purpose of this report is to explore header processing requirements for IP routers, identify different implementation alternatives and discuss their complexity. The conclusion of the study is that header processing can be implemented both in software and hardware, and the choice between software and hardware depends on requirements in terms of pricing and performance. Furthermore, we find that most IP header processing operations are relatively simple to implement, although packet classification (address lookup) needs special attention.

Internal Accession Date Only

*Swedish Institute of Computer Science, Box 1263, SE-164 29 Kista, Sweden
†Department of Computer Science, Lulea University of Technology, Lulea, Sweden
©Copyright Hewlett-Packard Company 1998

Header Processing Requirements and Implementation Complexity for IPv4 Routers

Andreas Moestedt, Torsten Köhler, Peter Sjödin

1997-10-13

1 Introduction

The distinction between routers and switches is getting more and more blurred. Routers are becoming simpler and cheaper, competing with LAN switches and customer premises equipment such as ISDN-to-Ethernet bridges. At the same time, router vendors are also developing high-end routers for the backbone, which has previously been regarded as a switching market for ATM and SDH equipment. In addition, recent inventions such as tag switching, IP switching, etc., are further confusing the distinction between routing and switching.

However, from a technical point of view, the main difference between routers and a packet switches of other kinds lies in the packet header processing that needs to be done at the input stage. A router is a packet switch for datagram networks, which means that a router needs to process the header of every packet in detail. A switch, at least in theory, does not need to perform the same amount of header processing.

The purpose of this report is to explore router input processing, in order to assess how much influence the header processing has on the complexity of routers. The report focuses on the implementation of the header-processing unit in an IP router, and the requirements that it must support. Both software and hardware solutions are considered, and their complexity and usage are discussed. In a router for gigabit speeds for example, a software-only solution is not

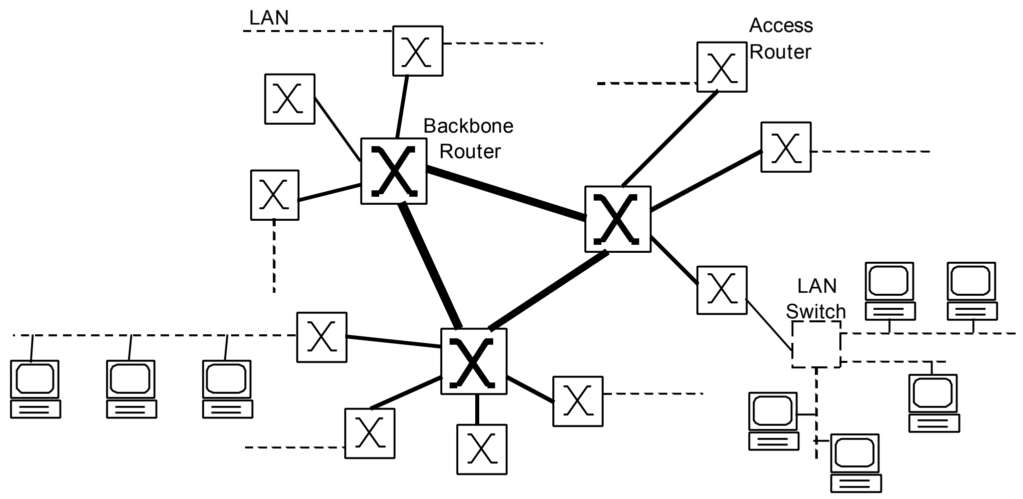

Figure 1. Example network architecture

likely to be a fast enough, whereas for a router on a LAN level it could be the only cost-effective alternative. The intention is to gain a better understanding of what task are necessary to perform when processing headers and how they can be implemented. To make a final decision on the software/hardware partitioning, the impact on the architecture from other activities in the router also has to be considered. This includes, for example, requirements on the packet buffers, packet scheduling, and switch arbitration.

1.1 Router Types

The technology discussion is held rather general, to keep it applicable on different kind of routers. There are mainly three kinds of routers considered: backbone routers, access routers, and IP aware LAN switches, seen in figure 1. The requirements for them are almost the same, but with a few differences due to their placement in the Internet network architecture. The backbone routers handle large amounts of aggregated IP traffic, at very high speeds. The access routers can be seen as gateways to the Internet, where packets are classified, bandwidth reservation is handled, and the traffic is policed to ensure that flows do not violate their assigned bandwidths. The IP aware LAN switch uses IP addresses to switch the packets on to, and between, different local area networks. By using IP addressing, the underlying protocols of the connected LANs are less important. IP technology also simplifies Intranet solutions and integration with the Internet.

1.2 Outline of the Document

In section 2 the overall design of the input part of a router is presented. The requirements on routers, according to RFC 1812, are summarized together with short corresponding discussions on implementation in section 3. The requirements are divided into five subsections, dealing with different aspects of the router. These sections are *packet header validation, IP header changes* (IP options excluded)*, source and destination verification, IP options* and *packet forwarding*. Section 4 discusses some software design issues, and the limits of a software implementation. Section 5 briefly deals with some aspects of a hardware implementation.

2 Router Overview

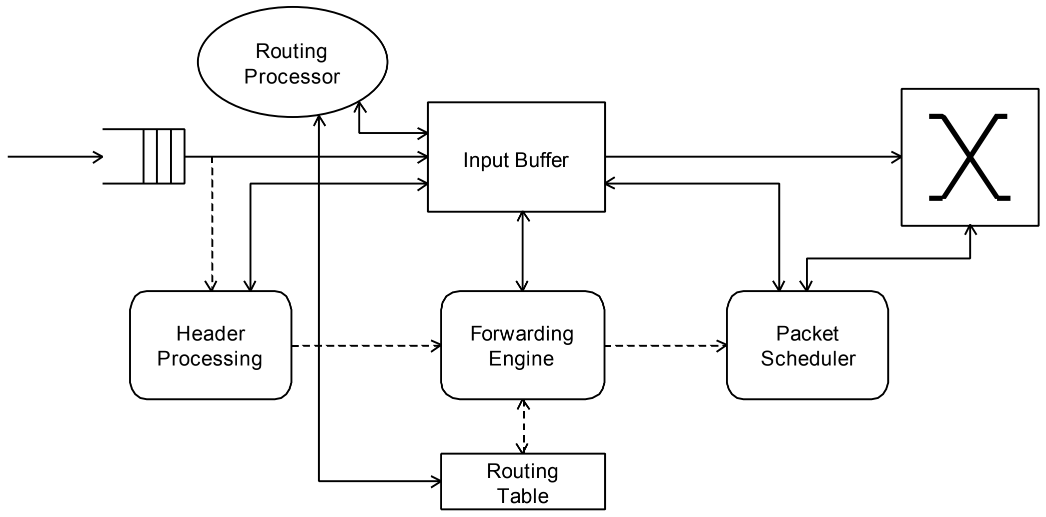

Figure 2 The main parts of a router

In a typical router design, six main parts can be identified on the input side. First is a *header processing unit* that checks the fields in the IP header, ensuring that they all have valid values. It also calculates the checksum, changes the TTL, and checks for options. The second part is an *input buffer* where packets are stored until they are sent to the output.

Destination addresses are looked up by a *forwarding engine* that decides the next-hop in the network, where the packet is to be sent. The forwarding engine uses a *forwarding table*, with all currently active routes. The fifth part of the router inport is a *routing processor*. It handles all routing updates and control information, and is responsible for keeping the forwarding table up to date. The last part is a *packet scheduler*, deciding what packets to send to the outports, dealing with flow management and QoS.

One of the basic design decisions related to input processing has to do with how the header-processing unit gets access to the packet header. One alternative is to write packets directly to the input buffer when packets enter the router, and let the header processing unit read the header from the input buffer. This is a simple solution that has the advantage that the header processing time can be amortized over the whole packet time, lowering the speed of the logic. The drawbacks are that packets may have to be dropped if too many small packets arrive next to each other, and that the input buffer needs to be fast enough to allow the header processing unit to access the input buffer while packets are being stored or retrieved. A second alternative is to stream packets through the header-processing unit before they are stored in the buffer. Such solutions put fewer requirements on the speed of the input buffer, and allow arbitrary combinations of packet sizes. The disadvantage is that it requires all work to be done at line-speed, which may require complex logic. A combination of the two approaches is to write the header to a small, fast buffer and the data part of the packet to the larger input buffer.

In this paper we mainly focus on the implementation of, and the requirements on, the header processing unit. Also some requirements on the forwarding engine are discussed.

3 Header Operations and Processing Requirements

3.1 Verification of Packet Header

3.1.1 Requirements

The router MUST check the following ([1] 5.2.2):

1. The length reported from the Link Layer must be large enough to hold the minimum IP packet length.(20 bytes).
2. The IP header checksum MUST be correct.
3. The IP version number MUST be 4.
4. The IP header length value MUST be large enough to hold the minimum IP packet length. (20 bytes)
5. The IP total length field MUST be large enough to hold the specified IP header length.

3.1.2 Implementation

The requirements above are all simple to implement in software. However, since the whole packet header needs to be examined, all header fields need to be read into the processor. Therefore, as a first approximation, the speed of a software implementation is limited by the memory bandwidth of the processor. See Section 4 for a more detailed discussion about this.

Implementing the above requirements in hardware is rather straightforward. No. 1 and 3-5 are simple comparisons, easily done at high speed. No. 2 is slightly harder since all ten 16-bit words in the header have to be added together. For a router with a link speed of 1 Gb/s the data rate is about 60 million words per second. This requires fast adders, but by doing the checksum calculation during the entire packet processing time, slower components can be used. In [2] an efficient way of calculating the checksum in hardware is presented. This implementation uses a simple MACH-435 PLD running at 40MHz, and copes with a data rate of 1.26 Gb/s, equivalent to a maximum of 8 million packets per second.

A possible combination is to do the checksum calculation in hardware but handle the other tests in software. This can lead to a more cost-effective solution, with cheap processors and simple hardware support.

Packets with options can have any header length between 20 and 60 bytes, complicating the processing. Handling of these packets is discussed in section 3.4.

3.2 Simple Changes to the IP Header

3.2.1 Requirements

1. The TTL field MUST be decremented by at least one. ([1] 5.3.1)
2. The IP header checksum MUST be updated if any changes to the header are done.
3. The router MUST NOT modify packets destined for the router itself. ([1] 5.2.1)

3.2.2 Implementation

Requirement no. 1 and 2 can be fulfilled using incremental IP header checksum updating, which is allowed when only the TTL field in the header has changed. ([1] 4.2.2.5) This operation does not add much complexity to a software implementation, and is also suitable for a hardware implementation [3].

Requirement 3 is simple to fulfil even if the TTL and checksum are changed in all incoming packets. This is done by allowing packets destined for the router itself to have a TTL that is 1 step too small, and correct that when the packet is processed by router.

3.3 Verification of Source and Destination Addresses

3.3.1 Requirements

1. Packets received as Link Layer broadcast or multicast, MUST have an IP Multicast group as destination, to be forwarded. ([1] 5.3.4)
2. A router SHOULD NOT forward packets with the following IP source or destination address: ([1] 5.3.7)
   - 2.1. Address on network 127 (except over a loopback interface)
   - 2.2. The following IP source addresses are invalid: ([1] 5.3.7, 4.2.2.11)
     - 2.2.1. Non unicast addresses.
     - 2.2.2. Address where network part contains only 0’s or 1’s.
     - 2.2.3. Address where host part contains only 0’s or 1’s.
   - 2.3. The following IP destination addresses are invalid: ([1] 5.3.7, 4.2.2.11, 4.2.3.1)
     - 2.3.1. Class E addresses.
     - 2.3.2. Address where network part contains only 0’s or 1’s.
     - 2.3.3. Address where host part contains only 0’s.
   - 2.4. Note: Some of the listed invalid addresses can be used within a network.

3.3.2 Implementation

Detecting multicast addresses (requirement 1) is simple since the high-order 4 bits of a multicast address are always 1110, so only a four-bit comparator is needed. Requirement 2.1 and 2.2.1 are also simple comparisons. If software or hardware should be used mainly depends on how other parts are implemented.

One possibility for checking for illegal addresses is to store the illegal addresses in the routing table, and check validity simultaneously with the next-hop lookup. Having to check the source address in the routing table is not desired, however. This wastes lookup resources and should be avoided if possible. Requirements 2.2.2, 2.2.3, 2.3.2 and 2.3.3 are easy to check in the routing table if each interface has its own table. If not, the header processing unit must know the network number and the netmask of the interface, and compare all 32 bits.

One possibility for checking for illegal addresses is to store the illegal addresses in the forwarding table, and check validity simultaneously with the next-hop lookup. Then two lookups are performed for each packet; one to validate the destination address and obtain the next hop address, and one to validate the source address. However, an address lookup is a complex operation and performing two lookups per packet is likely to be much more expensive than just doing one.

Requirements 2.2.2, 2.2.3, 2.3.2 and 2.3.3 are, in the general case, very difficult to satisfy since the host and network parts of an IP address are not readily available. One could speculate that these requirements were formulated back in those days when addresses were divided into classes (A, B, C, etc) and the host and network parts could be identified by just examining the address. Now, with the CIDR address organization, an address lookup would be required in order to get the host and network parts. Even worse, for a router to be able to perform this check for any address, all network prefixes need to be present in the routing table, which would defy the very purpose of using CIDR to aggregate addresses. Thus, the conclusion must be that these requirements can be ignored in the general case.

For addresses that concern the links that are directly attached to the router, it is, however, possible to validate the host and network parts of the address. This validation can be made by preloading the network number and the network mask of the interface and perform a simple 32-bit comparison (i.e. address filtering) This kind of checks are usually performed at places such as at the gateway between a private network and the public Internet.

It could be argued that many of the statements in requirement 2 are less important for a backbone router, especially since the requirements are classified as “SHOULD” and not “MUST”. The responsibility of checking them is then moved to the access routers.

3.4 Handling IP Options

3.4.1 Requirements

1 A router MUST support the source route options in forwarded packets. ([1] 5.3.13.4)

2 A router MUST support the Record Route option in forwarded packets. ([1] 5.3.13.5)

3 A router MUST support the timestamp option in forwarded packets. ([1] 5.3.13.6)

4 A router MUST NOT forward packets with more than one source route option. ([1] 5.2.4.1)

5 Unrecognized IP options MUST be passed through unchanged. ([1] 5.3.13.1)

3.4.2 Implementation

IP options can be processed in hardware [5], but since only a small fraction of all packets carry options, it is probably better to let a routing processor handle them. Furthermore, options are typically used for diagnostics and other types of network administration tasks, so the processing of options may not be very time-critical. There are several aspects of the IP options that complicates the processing, also favouring a software solution: there can be different options in a header in any order, the length of the option field is variable, 32-bit fields with addresses are only aligned on a byte boundary within the header. A simplifying restriction is that the maximum total length of a header is 60 bytes.

3.5 Forwarding

3.5.1 Requirements

1. Routers SHOULD be able to drop packets received on other interfaces than the one that packets to the source should be forwarded on. ([1] 5.3.8)
2. A router SHOULD NOT forward packets with the Strict Source and Record Route option, unless the router is the destination. ([1] 5.2.2)
3. A router SHOULD provide the ability to selectively forward packets using: ([1] 5.3.9)
   - 3.1. An include list - message definitions to be forwarded
   - 3.2. An exclude list - message definitions not to be forwarded
   - 3.3. A message definition specifies Source and destination prefixes, port number, protocol type, etc.
4. A router SHOULD implement precedence-ordered queue service. ([1] 5.3.3.1)
5. A router SHOULD consider the TOS field when deciding where to forward a packet to. ([1] 5.3.2)

3.5.2 Implementation

Meeting requirement 1 is simple, but a drawback is the need of doing a routing table lookup of the source address. Requirement 2 is also easy to handle, especially if packets carrying options are handled by a separate processor, as discussed in section 3.4.

Requirement 3 could to some extent be implemented using the routing table, more specifically by using flows identified by destination, port and protocol type. It is also possible to selectively handle packets to or from a specific network, by using the routing table. Flows specified as combinations of both source and destination prefixes, are difficult to identify using a conventional routing table. How to handle such flows should be inverstigated further.

Points 4 and 5 are related to provision of differentiated quality of service, which is not consider in this report. It is the packet scheduler that is responsible for meeting the last requirements, 4 and 5. Since no work has been done on packet scheduling or QoS, this will not be discussed here. How important these two requirements are, should be checked.

4 Comments on Software Implementation

4.1 Description

In a software design, the functionality can be divided into three basic blocks: *header processing*, *address lookup*, and *route handling*. These blocks can then be mapped on one or more processors, depending on the performance wanted. The discussion below focuses on one processor per task, but if lower performance is acceptable, one processor can handle more things.

4.2 Header Processing

The header-processing block contains header checksum validation, TTL check and calculation of the new checksum. It also ensures that some of the forwarding requirements are met.

All the headers must be read into the processor, which limits the speed of a software implementation. A low-cost Pentium® processor system has a typical maximum memory bandwidth of 80 MB/s [6], where approximately 60 MB/s could be utilized for header checking. If standard PC hardware is to be used, getting the headers into the main memory from, for example, a PCI card reduces the obtainable bandwidth by a factor of 2. This results in about 1-2 million packets per second. High-end systems have somewhat higher memory bandwidth; e.g. the Sun UltraSPARC architecture has a peak memory bandwidth of 170 MB/s.

On high-performance processors, doing the calculation while the (non-blocking) reads from memory complete can hide much of the memory latency. But it is mainly the speed of the memory system that limits the bandwidth, and not the speed of the processor. If the memory is fast enough, slower and cheaper processors can be used. If a specialized design with embedded processors is chosen, the memory system can be designed to better handle the high bandwidth, yielding a cost effective solution.

One interesting approach to get high performance using standard PC hardware is to exploit the support for high-speed graphics in newer processor systems. The Pentium® II has an advanced graphic port (AGP) that can be used for fast data transfers. The current AGP implementation has a peek bandwidth of 528 MB/s between the AGP port and the system chipset. The system chipset has the same high bandwidth to both the processor and to the main memory. If, and how, a fast router interface could be connected to the AGP port, and in that case how high bandwidth could be reached, needs to be investigated further.

The performance approximation above was obtained from a C program doing all the header checking. It ran as a user process on a Pentium (133MHz) based UNIX system, reaching a bandwidth of 30 MB/s. The same program without any header checking, with only reads from memory, reached 45 MB/s. Higher performance is likely to be achieved by rewriting the program in assembler. In that case computation and memory accesses can be interleaved more efficiently, leading to the bandwidth approximation of 60 MB/s on the system with a peak of 80MB/s.

4.3 Address Lookup

The main task of the address lookup is to find the next hop address for a packet, based on its destination IP address. Doing the lookup in software is feasible unless very high performance is needed. On a Pentium Pro system, 2 million forwarding lookups per second can be achieved [7]. For higher speed, a hardware solution is needed.

4.4 Route Handling

The route handling block supports the routing protocols, and builds the forwarding tables used in the router. It also handles packet with options, moving this complexity away from the fast path. The route handling is typically run on a separate processor.

5 Hardware Implementation

5.1 General

There are typically two cases when header processing in hardware is preferable. First, if the design is simple enough, it might be the most cost-effective way. Second it is the only choice at speeds where software cannot keep up. Independent of the speed, the input design where headers are streamed through the processing unit is probably the one to chose. This makes the memory interface simpler, and guarantees line-speed processing. As section 3 has shown, implementing header processing in hardware should be straightforward, supported by software to do less critical operations such as option processing. Since the design of a hardware solution is strongly dependent on what performance is needed, implementation complexity is difficult to assess before a target platform is chosen. We believe, however, that a few gate-arrays are enough to implement most of the processing task, up to rather high line speeds, at least a few Gb/s. The address lookup can also be done in hardware without being too complex, and reach very high speeds. With SRAM and a few simple gate-arrays, rates up to 10-50 million lookups per second is possible.

5.2 A Dataflow Approach

For the following discussion it is assumed that header processing for the common case is performed in hardware for the purpose of achieving high forwarding throughput. Deviations from the common case, such as option processing, fragmentation or error handling, are handled in software. Figure 2 shows a data flow graph of the header processing steps as defined in the previous sections. The processing steps are scheduled to be performed as soon as possible when the required input is available; time increases from left to right in the diagram. In case any one of the processing steps fails, the header processing can be aborted and the packet has to enter the slow path. Not included in the dataflow graph are all destination address checks that are assumed to take place in combination with the destination address lookup.

It is noteworthy that already with word 2 of the header available, the TTL and the header checksum field can be updated - even before the checksum has been verified.

What the graph also shows is that much of the processing can be done in parallel. It is only the horizontal flow that is sequential, and these four steps can work pipelined, as the header is streamed through the unit. This facilitates keeping up with fast line speeds.

6 Summary

In this paper we have explored the input processing that needs to be performed by an IP router. We have discussed how the different functions can be implemented and the complexity of the implementation alternatives. We have identified two main design alternatives that concern how the packet header is accessed by the header-processing unit: through streaming and through the input buffer. Streaming seems to give a more efficient implementation but requires that header processing is implemented in hardware. If a software implementation is used, input buffer header access is the only alternative.

We have considered implementing all header-processing functions in software, and concluded that the processing complexity for this would be low. Instead we think that the memory bandwidth of the processor is the main determining factor for performance of a software implementation.

For a hardware-based solution, we think that most functions are possible to implement with standard components (e.g. RAM and simple gate arrays). Some functions are still best to implement in software, such as option processing, but we think that this is justifiable since packets with options in many cases are not urgent to process, and therefore could be handled by a slower, central control processor.

We have not found any single header processing function that is very complex to implement. Furthermore, the number of functions that need to be implemented is rather modest, and the degree of dependencies between the functions is low. These things in combination indicate, in our opinion, that header processing in routers need not be significantly more complex than header processing in other types of packet switches.

7 References

[1] Baker, F., “Requirements for IP Version 4 Routers,” Network Working Group RFC-1812, Cisco Systems, June 1996.

[2] Touch, J., and Parham, B., ”Implementing the Internet Checksum in Hardware,” Network Working Group RFC-1936, ISI, April 1996.

[3] Rijsinghani, A., ”Computation of the Internet Checksum via Incremental Update,” Network Working Group RFC-1624, Digital Equipment Corp., May 1994.

[4] Postel, J., “Internet Protocol,” STD 5, RFC-791, USC/Information Sciences Institute, September 1981.

[5] Tantawy, A., Koufopavlou, O., Zitterbart, M., and Abler, J., “On the Design of a Multigigabit IP Router,” Journal of High Speed Networks. Vol. 3, no. 3, 1994.

[6] URL: http://www.cs.virginia.edu/stream/standard/Bandwidth.html

[7] Degermark, M., Brodnik, A., Carlsson, S., and Pink, S., “Small Forwarding Tables for Fast Routing Lookups,” Proceedings of SIGCOMM’97, pp. 3-14, 1997.

Title:

Creator:

Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

Figure 3 The dataflow during header processing